1 Parallel Semidefinite Programming and Combinatorial Optimization

STEVEN J. BENSON
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

1.1 INTRODUCTION

The use of semidefinite programming in combinatorial optimization continues to grow. This growth can be attributed to at least three factors: new semidefinite relaxations that provide tractable bounds to hard combinatorial problems, algorithmic advances in the solution of semidefinite programs (SDP), and the emergence of parallel computing.

Solution techniques for minimizing combinatorial problems often involve approximating the convex hull of the solution set and establishing lower bounds on the solution. Polyhedral approximations that use linear programming (LP) methodologies have been successful for many combinatorial problems, but they have not been as successful for problems such as maximum cut and maximum stable set. The semidefinite approximation for the stable set problem was proposed by Grötschel, Lovász, and Shrijver [32] and further developed by Alizadeh [1], Polijak, Rendl, and Wolkowicz [48], and many other authors. The Lovász number [43] is the solution of an SDP that provides an upper bound to the maximum clique of a graph and a lower bound to its chromatic number. Tractable bounds have also been provided for MAX-CUT [30], graph bisection [47], MAX k-CUT [27, 24], graph coloring [33], and the quadratic assignment problem [61]. Many more combinatorial problems can be relaxed into a semidefinite program, and some of these relaxations offer a projection back to a feasible solution that is guaranteed to be within a specified fraction of optimality. These combinatorial problems are NP hard in the general case, so even approximate solutions are difficult to find. Most polyhedral relaxations do not offer a performance guarantee, but Geomans and

Williamson [30], in a now classic result, also proved a polynomial-time approximation algorithm for the MAX-CUT problems with strong performance guarantees. Other theoretical results on the effectiveness of semidefinite relaxation have been developed for MAX-SAT [2, 3], MAX-2-SAT [55], MAX-(2+p)-SAT [25], and MAX k-CUT [27], and graph coloring [37]. See Laurent and Rendl [42] for additional relationships between combinatorial optimization and semidefinite programming.

In semidefinite programming, the variable is not a vector but a symmetric matrix. Semidefinite problems minimize a linear function of the matrix subject to linear constraints and the essential constraint that the matrix be positive semidefinite. The last constraint is nonlinear and nonsmooth, but convex, so semidefinite programs are convex optimization problems. Many of the algorithmic ideas for optimizing over a polyhedral set have been extending to semidefinite optimization. Interior-point solvers, in particular, have polynomial complexity and have been used to solve broad classes of SDP [10, 17, 50, 58, 53]. Other methods, such as a generalized penalty method [39], the low-rank factorization method [20], and a spectral bundle method [34], have also proven effective for combinatorial problems. Surveys by Todd [51] and Wolkowicz [56] present many examples of SDP and the algorithms most frequently used for solving them.

Although interior-point methods have proven reliable on small and medium-sized semidefinite programs, the computational and storage demands of these methods can easily exhaust the resources of most computers and limit the size of problems that can be solved. Although they are "polynomially" solvable, semidefinite programs with dimension above 10,000 have been extremely hard to solve in practice. Graphs with more vertices and edges routinely arise in VLSI design and other applications. Much of the research in this field focuses on solving medium-scale problems more quickly and on solving large-scale problems by any means possible. High-performance computers have become more common, and the additional memory and processing power have alleviated some of the bottlenecks in large-scale semidefinite programs. The MPI [31] standard for passing messages between processors has facilitated software development on these platforms. Parallel linear solvers [6, 15] and nonlinear solvers [8, 36] have been developed using MPI. At least five parallel solvers for semidefinite optimization have also been written for high-performance architectures [7, 17, 59, 46, 54]. This chapter will use one of those solvers, PDSDP, for its computations. All computations in this chapter used multiple processors on a cluster of 360 2.4 GHz Pentium Xeon processors. Each processor had at least 1 GB of RAM, and they were connected by a Myrinet 2000 network.

1.2 MAXIMUM-CUT PROBLEMS

Let there be an undirected and connected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ and $E \subset \{(i, j) : 1 \leq i < j \leq n\}$. Let the edge weights $w_{i,j} = w_{j,i}$ be given such that $w_{i,j} = 0$ for $(i, j) \notin E$, and in particular, let $w_{i,i} = 0$. The maximum cut problem (MAX-CUT) is to find a partition (V_1, V_2) of V so that the sum of the edge weights between V_1 and V_2 is maximized. It is well known that MAX-CUT can be formulated as

$$\text{maximize} \quad \frac{1}{2} \sum_{i<j} w_{i,j} \, (1 - v_i v_j) \quad \text{subject to} \quad v_i \in \{-1, 1\}, \; i = 1, \ldots, n.$$

Defining

$$C_{i,j} = \begin{cases} -\frac{1}{4} w_{i,j} & \text{if } i \neq j \\ \frac{1}{4} \sum_{k=1}^{n} w_{i,k} & \text{if } i = j \end{cases},$$

one can rewrite as:

$$\text{maximize} \quad \sum_{i,j} C_{i,j} v_i v_j \quad \text{subject to} \quad v_i v_i = 1, \; i = 1, \ldots, n.$$

If we introduce a matrix X from the set of symmetric $n \times n$ matrices, denoted $\mathbb{S}^n$, the combinatorial problem can be relaxed to

$$\text{maximize} \quad \sum_{i,j} C_{i,j} X_{i,j} \quad \text{subject to} \quad X_{i,i} = 1, \; i = 1, \ldots, n, \quad X \succeq 0. \quad (MC)$$

The notation $X \succeq 0$ means that $X \in \mathbb{S}$ and positive semidefinite. The formulation (MC) is equivalent to MAX-CUT when X has the form $X_{i,j} = v_i v_j$, but the semidefinite relaxation ignores this symmetric rank-one constraint on the matrix. The objective function and equality constraints in (MC) are both a linear function of X, and the feasible set is convex.

The work of Geomans and Williamson replaced the scalars v_i with unit vectors $v_i \in \mathbb{R}^n$ and scalar products $v_i v_j$ with vector inner products $v_i^T v_j$. A solution $(v_1, \ldots, v_n)$ consists of n points on the surface of the unit sphere in $\mathbb{R}^n$, each representing a node in the graph. After solving (MC), their algorithm partitions the unit sphere into two half-spheres using a random vector from the same sphere. The algorithm forms a partition consisting of the v_i in each half-sphere. Furthermore, Goemans and Williamson established the celebrated result that if all the weights are nonnegative, the expected value of such randomly generated cuts is at least 0.878 times the maximum cut value. This result gives a strong performance guarantee not available from polyhedral relaxations.

Similar problems impose additional constraints on the feasible set. For example, vertices s and t may be forced in different subsets by relaxing the

constraint that $v_s + v_t = 0$ into $X_{s,s} + X_{s,t} + X_{t,s} + X_{t,t} = 0$. The minimum bisection problem partitions the vertices such the V_1 and V_2 have the same cardinality. The semidefinite relaxation of the constraint that $|\sum v_i| = 0$ is $\sum X_{i,j} = 0$. Randomized procedures for the $s - t$ cut problem and minimum bisection problem have also been effective [11].

1.3 COLORS, CLIQUES, AND STABLE SETS

Given an undirected graph $G = (V, E)$, a *clique* of graph G is a subset set of vertices such that every pair of vertices is adjacent. We denote the cardinality of the largest clique in a graph as $\mathcal{C}(G)$. A *vertex coloring* of a graph is an assignment of colors to the vertices V such that no two adjacent vertices receive the same color. A graph is *k-colorable* if it can be colored with k colors or fewer. The smallest number of colors needed for this coloring is called the *chromatic number* of G, which we denote as $\mathcal{X}(G)$.

Aside from its theoretical interest, the maximum clique problem arises in applications in information retrieval, experimental design, signal transmission, and computer vision [5]. Graph coloring arises when finite differences are used to approximate sparse Hessian matrices, and well as in applications in computer register allocation [19, 22, 21], timetable scheduling [13, 26, 57], and electronic bandwidth allocation [28].

These classic problems in combinatorial optimization are well known to be NP-hard [29]. The maximum clique problem and minimum coloring problem can be solved by using polynomial-time algorithms for special classes of graphs such as perfect graphs and t-perfect graphs, circle graphs and their complements, circular arc graphs and their complements, claw-free graphs, and graphs with long odd cycles [44], but the existence of a polynomial time algorithm for arbitrary graphs seems unlikely.

Since the coloring of every vertex in a clique must be different, the cardinality of any clique in G is a lower bound on the chromatic number of G. The inequality $\mathcal{C}(G) \leq \mathcal{X}(G)$ is true for all graphs. When a graph (and every node induced subgraph) has a chromatic number that equals the cardinality of the largest clique, it is known as a *perfect graph*. For this special class of graphs, the maximum clique and vertex coloring problems can be solved to optimality by using a polynomial algorithm.

For a graph with n vertices, the Lovász number, $\vartheta(G)$, is the solution to a semidefinite program whose variable X is a symmetric $n \times n$ matrix. The SDP is

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i,j} X_{i,j} \\ \\ \text{subject to} & X_{i,j} = X_{j,i} = 0, \quad \forall\, (i,j) \notin \mathrm{E} \\ & \text{trace}\,(X) = 1, \quad X \succeq 0. \end{array} \qquad (LOV)$$

This number satisfies the inequality $\mathcal{C}(G) \leq \vartheta(G) \leq \mathcal{X}(G)$ and provides a bound to both the maximum clique problem and minimum graph coloring problem.

Equivalent to the maximum clique problem are two other combinatorial problems. A *stable set* of vertices (or *vertex packing* or *independent set*) is a subset of V such that no two vertices are adjacent. A *vertex cover* is a subset of vertices that are incident to each edge in the graph. With $\bar{G}$ denoted as the graph complement of G, the following statements concerning any $S \subset V$ are known to be equivalent:

1. S is a clique of G,
2. S is a stable set of $\bar{G}$,
3. $V \setminus S$ is vertex cover of $\bar{G}$.

Accordingly, the problems of finding a maximum stable set of $\bar{G}$, a maximum clique in G, and a minimum vertex cover in $\bar{G}$ are equivalent. The *maximum stable set problem* (MSS) asks for the stable set with the maximum cardinality. Bomze et. al. [16] provide a history of results relating to this problem.

Much like the formulation of Kleinberg and Goemans [38] the SDP relaxation of the MSS problem will assign each vertex an integer value of ± 1. One of the two sets will be a stable set. Given a graph G with n vertices, one formulation adds an artificial vertex v_{n+1} with no edges connecting it to other vertices. Since the artificial vertex is obviously a member of the maximal stable set of the new graph, its sign will used to identify the stable set and enforce the constraints of the problem. The MSS problem can be stated as:

$$\begin{array}{lrcll} \text{maximize} & \frac{1}{2}\sum_{i=1}^{n}\left(v_i^2 + v_i v_{n+1}\right) & & & \\ & & & & (MSS) \\ \text{subject to} & |v_i + v_j + v_{n+1}| & = & 1 & \forall\ (i,j) \in \mathrm{E}, \\ & v_i & \in & \{-1,1\} & i = 1,\ldots,n+1, \end{array}$$

The semidefinite relaxation of MSS introduces a symmetric matrix $X \in \mathbb{S}^{n+1}$ and sets $X_{i,j} = v_i v_j$. Since $|v_i + v_j + v_{n+1}| = (v_i + v_j + v_{n+1})^2$, the semidefinite relaxation is

$$\begin{array}{lrcll} \text{maximize} & \frac{1}{2}\sum_{i=1}^{n} X_{i,i} + X_{i,n+1} & & & \\ \text{subject to} & \sum_{i,j \in \{s,t,n+1\}} X_{i,j} & = & 1 & \forall\ (s,t) \in \mathrm{E} \\ & X_{i,i} & = & 1 & i = 1,\ldots,n+1, \quad X \succeq 0. \end{array}$$

Clearly, there are nine variables in each equation corresponding to an edge, and one variable in the other equations. Imposing the additional constraint

that the matrix X be a rank-one matrix of the form $X = vv^T$ would make it equivalent to (MSS). Relaxing this constraint to include all symmetric positive semidefinite matrices makes the feasible region convex, and the solution to this problem provides an upper bound to the integer program (MSS).

In order to favor the inclusion of selected vertices into the stable set, the *weighted maximal stable set* problem has a similar formulation. Given weights w_i on the vertices, this problem seeks to maximize

$$\frac{1}{2}\sum_{i=1}^{n} w_i \left(v_i^2 + v_i v_{n+1}\right)$$

subject to the same constraints as (MSS). The problems can also be addressed by using the semidefinite relaxation. In polyhedral relaxations of the maximal stable set problem, utilizing larger cliques is crucial for a tight approximation to the convex hull of the integer program. These cliques can also improve the semidefinite relaxation. Given cliques $\mathcal{C}^1, \ldots, \mathcal{C}^d$, such that $\mathcal{C}^k$ has n_k vertices, stable sets $v \in \{-1, 1\}^n$ must satisfy

$$|(n_k - 1)v_{n+1} + \sum_{v_i \in \mathcal{C}^k} v_i| = 1$$

for $k = 1, \ldots, d$. This formulation has a semidefinite relaxation that more closely approximates the convex hull of the integer program.

The semidefinite relaxation of the graph coloring problem is similar to that of the maximum cut problem. Instead of assigning colors or integers to the vertices of the graph, a unit vector $v_i \in \mathbb{R}^n$ is assigned to the each of the n vertices i in V. To capture the property of coloring, the vectors of adjacent vertices should differ in a natural way. From the definition in [37], the vector k- coloring of G is an assignment of unit vectors $v_i \in \mathbb{R}^n$ to each vertex i in V such that for any two adjacent vertices i and j, the dot product of the vectors satisfies the inequality $v_i^T v_j \leq -\frac{1}{k-1}$. In other words, the angle between the vectors of adjacent vertices must be sufficiently large. Define the matrix V such that column i is given by v_i, and let $X = V^T V$. The matrix X is positive semidefinite and satisfies the inequalities $X_{ij} = X_{ji} \leq -\frac{1}{k-1}$ for each pair of adjacent edges (i, j). Obviously, any matrix is n-colorable, so the graph coloring problem can be posed as

$$\begin{array}{lllll} \text{Minimize} & \text{rank}(X) & & & \\ \text{Subject to} & X_{ij} + X_{j,i} & \leq & -\frac{2}{n-1} & \text{for } (i,j) \in \text{E} \quad \text{(COL)} \\ & X_{i,i} & = & 1 & i = 1, \ldots, n. \end{array}$$

A semidefinite relaxation of the graph k-coloring problem can be written by replacing the n with a k. Ignoring the objective function, the problem is now a semidefinite program that seeks to find a feasible point.

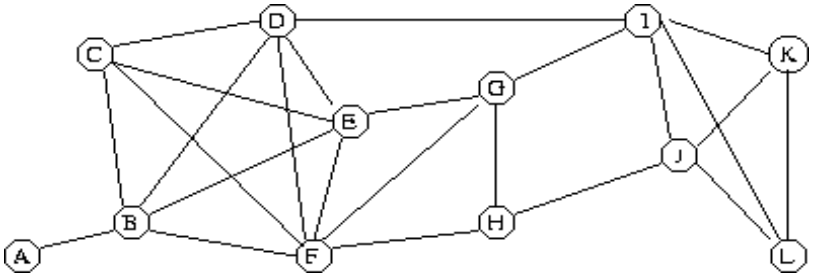


Fig. 1.1 Sample graph with 12 vertices and 23 edges.

Figure 1.1 shows a small graph to illustrate these combinatorial problems. In this graph, $\{A, C, F, H, I, K\}$ and $\{B, D, F, G, J, L\}$ form a maximum cut; $\{B, C, D, E, F\}$ is the maximum clique; $\{A, C, I, H, L\}$ is the maximum stable set; and $\{B, H, I\}$,$\{C, G, J\}$,$\{D, K\}$,$\{E, L\}$,$\{A, F\}$ form an optimal coloring.

1.4 AN INTERIOR-POINT ALGORITHM

The inner product in the space $\mathbb{S}^n$ is denoted by $\bullet$, which is defined as $U \bullet V = \sum_{i,j=1}^{n} U_{i,j} V_{i,j}$. Given input data $C, A_1, \ldots, A_m \in \mathbb{S}^n$ and scalars $b_1, \ldots b_m$, we state the pair of semidefinite programs

$$\begin{array}{lllll} (P) & \inf\ C \bullet X & \text{subject to} & A_i \bullet X = b_i,\ i = 1, \ldots, m, & X \succeq 0, \\ (D) & \sup \sum_{i=1}^{m} b_i y_i & \text{subject to} & \sum_{i=1}^{m} A_i y_i + S = C, & S \succeq 0. \end{array}$$

In this form, (P) is referred to as the *primal* problem whereas (D) is referred to as the *dual* problem. Variables X and (y, S) are called feasible solutions to (P) and (D) if they satisfy the constraints in their respective problems. Interior feasible points are feasible solutions such that $X \succ 0$ and $S \succ 0$. The notation $X \succ 0$ for $X \in \mathbb{S}^n$ means that X is positive definite. The interior feasible sets of (P) and (D) will be denoted by $\mathcal{F}^0(P)$ and $\mathcal{F}^0(D)$, respectively. A well-known duality theorem states that provided there is a strictly interior point to (P) and (D), there exist primal and dual optimal solutions, (X^*, y^*, S^*) and $C \bullet X^* = b^T y^*$.

This discussion also assumes that the A_is are linearly independent, there exists $X \in \mathcal{F}^0(P)$, and $(y, S) \in \mathcal{F}^0(D)$. Optimal solutions X^* and (y^*, S^*) are also characterized by the equivalent conditions that the duality gap $X^* \bullet S^*$ is zero and the product $X^* S^*$ is zero. Moreover, for every $\nu > 0$, there exists a unique primal-dual feasible solution (X_ν, y_ν, S_ν) that satisfies the perturbed optimality equation $X_\nu S_\nu = \nu I$. The set of all solutions $\mathcal{C} \equiv \{(X_\nu, y_\nu, S_\nu) : \nu > 0\}$ is known as the central path, and $\mathcal{C}$ serves as the ba-

sis for path-following algorithms that solve (P) and (D). These algorithms construct a sequence $\{(X, y, S)\} \subset \mathcal{F}^0(P) \times \mathcal{F}^0(D)$ in a neighborhood of the central path such that the duality gap $X \bullet S$ goes to zero. A scaled measure of the duality gap that proves useful in the presentation and analysis of path-following algorithms is $\mu(X, S) = X \bullet S/n$ for all $(X, S) \in \mathbb{S}^n \times \mathbb{S}^n$. Note that for $X \succ 0$, $S \succ 0$, we have $\mu(X, S) > 0$ unless $XS = 0$. Moreover, $\mu(X_\nu, S_\nu) = \nu$ for all points (X_ν, y_ν, S_ν) on the central path.

Several polynomial algorithms can solve a pair of semidefinite programs. Helmberg-Rendl-Vanderbei-Wolkowicz/Kojima-Shida-Hara/Monteiro, Nesterov-Todd (see Monteiro [45] and references therein). All these algorithms possess $O(\sqrt{n}\log(1/\epsilon))$ iteration complexity to yield accuracy ϵ. This section summarizes the dual-scaling algorithm for solving (P) and (D). For simplicity, the discussion assumes that the matrix variables are a single semidefinite variable, but the extension of the algorithm to direct products of semidefinite matrices is relatively simple.

Let the symbol $\mathcal{A}$ denote the linear map $\mathcal{A} : V \to \mathbb{R}^m$ defined by $(\mathcal{A}X)_i = \langle A_i, X\rangle$; its adjoint $\mathcal{A}^T : \mathbb{R}^m \to V$ is defined by $\mathcal{A}^* y = \sum_{i=1}^m y_i A_i$. The dual-scaling algorithm applies Newton's method to $\mathcal{A}X = b$, $\mathcal{A}^T y + S = C$, and $X = \nu S^{-1}$ to generate

$$\begin{aligned} \mathcal{A}(X + \Delta X) &= b, & (1.1)\\ \mathcal{A}^T(\Delta y) + \Delta S &= 0, & (1.2)\\ \nu S^{-1}\Delta S S^{-1} + \Delta X &= \nu S^{-1} - X. & (1.3) \end{aligned}$$

Equations (1.1 – 1.3) are referred to as the Newton equations; their Schur complement is

$$\nu \begin{pmatrix} A_1 \bullet S^{-1}A_1S^{-1} & \cdots & A_1 \bullet S^{-1}A_mS^{-1} \\ \vdots & \ddots & \vdots \\ A_m \bullet S^{-1}A_1S^{-1} & \cdots & A_m \bullet S^{-1}A_mS^{-1} \end{pmatrix} \Delta y = b - \nu\mathcal{A}S^{-1}. \quad (1.4)$$

The left-hand side of this linear system is positive definite when $S \succ 0$. In this chapter, it will sometimes be referred to as M. DSDP computes $\Delta' y := M^{-1}b$ and $\Delta'' y := M^{-1}\mathcal{A}S^{-1}$. For any ν,

$$\Delta y := \frac{1}{\nu}\Delta' y - \Delta'' y$$

solves (1.4).

Using (1.2), (1.3), and Δy, we get

$$X(\nu) := \nu\left(S^{-1} + S^{-1}(\mathcal{A}^T\Delta y)S^{-1}\right),$$

which satisfies $\mathcal{A}X(\nu) = b$. Notice that $X(\nu) \succ 0$ if and only if

$$C - \mathcal{A}^T(y - \Delta y) \succ 0.$$

If $X(\nu) \succ 0$, a new upper bound

$$\bar{z} := C \bullet X(\nu) = b^T y + X(\nu) \bullet S = b^T y + \nu \left(\Delta y^T \mathcal{A} S^{-1} + n \right)$$

can be obtained without explicitly computing $X(\nu)$. The dual-scaling algorithm does not require $X(\nu)$ to compute the step direction defined by (1.4); indeed, the solvers will not compute $X(\nu)$ unless specifically requested. This feature characterizes the algorithm and its performance.

For $\rho > n + \sqrt{n}$, either (y, S) or X reduces the dual potential function

$$\psi(y) := \rho \log(\bar{z} - b^T y) - \ln \det S$$

enough at each iteration to achieve linear convergence. More details about the algorithm and its implementation can be found in [10].

```
Choose an upper bound z̄ and y such that S ← C − A^T y ≻ 0.
for k ← 0, ..., convergence do
    Select ν.
    Compute M and AS^{-1}.
    Solve MΔ'y = b, MΔ''y = AS^{-1}.
    if C − A^T(y − Δy) ≻ 0 then
        z̄ ← b^T y + ν (Δy^T AS^{-1} + n).
        ȳ ← y, Δy ← Δy, μ̄ ← ν.
    end if
    Find α_d to reduce ψ, and set y ← y + α_d Δy, S ← C − A^T y.
end for
Optional: Compute X using ȳ, Δy, μ̄.
```

1.5 PARALLEL COMPUTING

Even relatively small graphs can lead to large SDP that are difficult to solve. The dense matrix M contains a row and column for each constraint. Although interior-point methods for semidefinite programming are computationally intensive, the high memory requirements are usually the bottleneck that restrict the size of problems that can be solved. A graph with n vertices, for instance, has a MAX-CUT relaxation such that the matrix M has n rows and columns. For the Lovász problem, the solver constructs M with dimension of $|\bar{E}| + 1$, where $|\bar{E}|$ is the number of edges in the complement of the graph, and for the MSS problem, $M \in \mathbb{S}^{|E|+n}$. Graphs with only few hundred vertices can easily contain thousands of edges. A dense matrix with $10,000$ rows and columns requires about 800 MB RAM, which can easily exhaust the resources of most serial architectures. The total memory requirements of the solver may be much greater.

The most computationally expensive tasks in each iteration of the dual-scaling algorithm are usually the construction of the linear system (1.4) and the factorization of M. As documented in [12], the worst-case computational complexity of computing M is $\mathcal{O}(m^2n^2 + mn^3)$, and factoring it is $\mathcal{O}(m^3)$. Sparsity in the data can reduce the cost of the former task, but its complexity remains much greater than interior-point iterations for linear or second-order cone programming.

In order to reduce the time needed to solve these applications and to satisfy the memory requirements of interior-point methods, parallel implementations distribute (1.4) over multiple processors and compute on it in parallel. Either shared-memory or distributed-memory paradigms can be used. Borchers [18] used Open MP directives and shared-memory implementations of BLAS and LAPACK to parallelize his primal-dual interior-point solver. Computational studies with sixteen processors showed overall parallel efficiency on large problems between 20% and 100%. The interior-point solvers of Yamashita et. al. [59] and PDSDP [7] assumed a distributed-memory paradigm and used MPI to pass messages between processors. Both solvers used the parallel Cholesky factorization in ScaLAPACK [15]. All these solvers gave each processor local access to the data A_i and C. Furthermore, all three solvers explicitly assigned to each processor a mutually exclusive set of rows in M. As illustrated in Figure 1.2, the solvers explicitly manage parallelism in M and have local access to the data matrices. In the distributed-memory solvers, each processor also has a local copy of the matrices X and S.

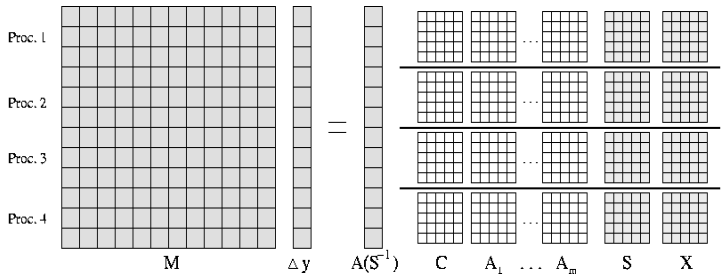


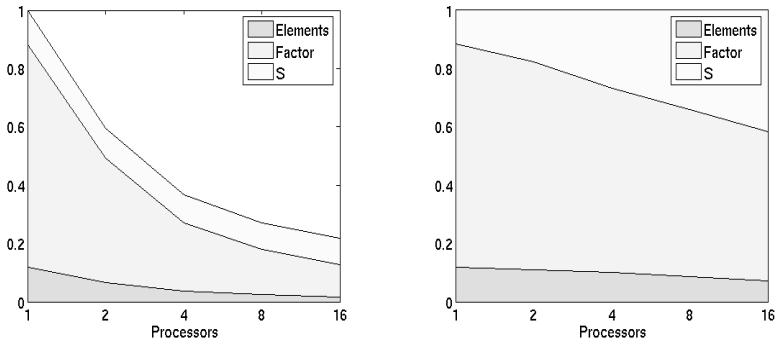
Fig. 1.2 Distribution of data structures and computations over multiple processors.

The DSDP [10] software package served as the basis for a parallel implementation of the dual-scaling algorithm. Since each element of M has the form $M_{i,j} = A_i \bullet (S^{-1}A_jS^{-1})$, each element in row (or column) j requires computing the matrix $S^{-1}A_jS^{-1}$. The cost of this product usually exceeds the cost of its inner product with the data matrices A_i. In order to reduce duplication of work, it is efficient for one processor to compute all elements of (1.4) in the same row. Inserting a row of elements into a parallel matrix structure may require sending some elements to other processors. The overhead of the message, however, is small compared to the upper limit of

$\mathcal{O}(n^3 + n^2 m)$ floating-point operations (flops) to compute the elements of the row. Although an interior-point iteration for semidefinite programming is more computationally intensive that a similar iteration for linear or second-order conic programming, the additional complexity allows for better parallel scalability because the ratio of floating-point operations to interprocessor messages is relatively high.

Load balancing among processors is trivial for most combinatorial applications. Since the data constraints in (MC) all have a single nonzero, the rows of M should be divided as evenly as possible over the processors. For (MSS) and (COL), there are two types of constraint matrices; in these examples, the rows corresponding to each type constraints are divided as evenly as possible. Only in (LOV) is there one constraint, (trace $(X) = 1$), whose structure is significantly different from the others. In this case, the identity matrix has more nonzeros and higher rank than do the other constraint matrices.

After factoring M and solving (1.4), PDSDP broadcasts the step direction Δy from a parallel vector structure to a serial vector structure on each processor. The serial vector structure contains all elements in Δy, and PDSDP uses it to calculate a suitable step-length, update the dual matrix S, factor it, and compute an appropriate barrier parameter ν for the next iteration.



(a) Time (b) Percentage

Fig. 1.3 Parallel scalability of computing M, factoring it, and other operations.

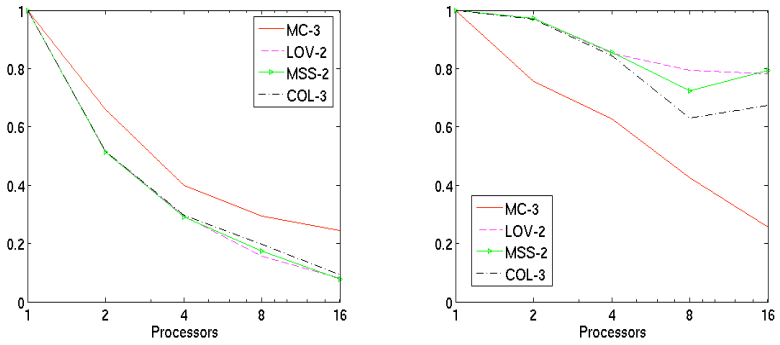
Most analysis of the scalability of interior-point algorithms for semidefinite programming has concerned three mutually exclusive parts of the algorithms: computing the elements of the Newton equations (Elements), factoring the Schur complement matrix (Factor), and other operations that consist primarily of computing X and factoring S (S). Using a Lovasz problem for a graph with 100 vertices and 990 edges as an example, Figure 1.3 shows the relative amount of time spent in each part of the algorithm for five groups of processors. Normalizing times such that one processor required one unit of time

to solve the problem, Figure 1.3(a) shows the time spent in each part of the algorithm for each group of processors. Figure 1.3(b) shows these times of the three parts as a percentage of the entire solution time. In this case, a single processor spent about 12% of its time computing M and 75% of the time factoring it. Using sixteen processors, about 8% of the time was spent computing the matrix and 50% of the time was spent factoring it. The remaining operations are mostly serial, so those times did not change significantly as the number of processors increased. On a single processor these operations amounted to about 10% of the time, and on sixteen processors they amounted to about 40% of the time.

A set of computational tests shown in Table 1.1 measure the overall parallel scalability and efficiency of the solver. on three instances of the semidefinite relaxations of (MC), (LOV), (MSS), and (COL). The graphs were chosen such that the SDP's could be solved on on single a single processor with 1 GB RAM and still scale well to additional processors. For the larger instances of (LOV), (MSS), and (COL), the overall parallel efficiency on sixteen processors was about 80%. The time spent factoring M dominated the overall solution time, and the scalability of ScaLAPACK heavily influenced the scalability of the entire solver. The overall efficiency on the maximum-cut relaxation was less because the semidefinite blocks have dimension $n = m$, and the factorization of S in serial uses a significant percentage of the solution time. Figure 1.4 shows the overall scalability of PDSDP on four instances of semidefinite programs.

Table 1.1 Scalability of PDSDP on semidefinite relaxations.

					Processors and Seconds				
Name	$\|V\|$	$\|E\|$	n	m	1	2	4	8	16
MC-1	1000	5909	1000	1000	46	30	16	11	8
MC-2	5000	14997	5000	5000	6977	4108	2592	2274	1460
MC-3	7000	17148	7000	7000	16030	10590	6392	4721	3928
LOV-1	100	2970	100	1981	43	23	13	8	4
LOV-2	150	3352	150	7824	3394	1747	995	535	271
LOV-3	150	7822	150	3354	196	104	49	34	17
MSS-1	100	2970	100	2081	261	141	75	55	31
MSS-2	150	3352	150	7974	22628	11630	6616	3916	1781
MSS-3	150	7822	150	3504	1671	875	494	307	159
COL-1	100	2970	100	3070	518	276	124	99	49
COL-2	150	3352	150	3502	770	407	186	157	72
COL-3	150	7822	150	7972	9445	4866	2796	1875	875



(a) Times. (b) Efficiency.

Fig. 1.4 Overall time and parallel efficiency on four SDP.

Distributing M across processors allows larger problems to be solved, but even this technique may not be sufficient for problems with hundreds of thousands of constraints. One idea has been to use the conjugate gradient method to solve (1.4). Since iterative linear solvers require only the product of the matrix with a vector, the entire matrix M does not have to be stored. Choi and Ye have used this approach on maximum cut problems [23] and Toh and Kojima [52] have explored similar ideas for primal-dual methods in conjunction with a reduced space preconditioner. Another approach for extraordinarily large problems is to use methods such as low-rank factorizations or the parallel bundle method [54].

1.6 FEASIBLE SOLUTIONS

In their randomized algorithm, Goemans and Williamson defined $V \in \mathbb{R}^{n\times n}$ such that $X = VV^T$ and selected a vector $u \in \mathbb{R}^n$ from a uniform distribution of unit vectors. Then they let

$$\hat{v} = Vu$$

and defined

$$v_i = sign(\hat{v}_i),\ i = 1, \ldots, n.$$

They proved that if all the edge weights $w_{i,j} \geq 0$, the cut generated by the randomized algorithm has an expected value greater than 0.878 of optimality.

For another measure of optimality, let $q(v) := \sum_{i,j} C_{i,j} v_i v_j$, and let $\overline{q}$ and $\underline{q}$ be the upper and lower bound of $q(v)$. Furthermore, let $E[\cdot]$ be the expectation operator. Nesterov [47] proved that for arbitrary adjacency matrices, the

randomized technique above gives an approximate solution v such that

$$E\left[\frac{q(v)-\underline{q}}{\overline{q}-\underline{q}}\right] \geq \frac{4}{7}.$$

As the number of applications of this randomized algorithm increases to infinity, the probability of generating a solution whose objective value equals or exceeds the expectation increases to one. In practice, this algorithm is very quick and can be repeated many times.

Table 1.2 Approximate Maximum Cuts from the Semidefinite Relaxation.

		Objective Values				Seconds	
Name	n	SDP	Rand	Ratio	$\|P\|$	SDP	Rand
MC1a	1000	29,428.5	28,158	0.957	4	66	11
MC1b	1000	4,737.8	3,419	0.722	4	70	11
MC3a	3000	30,226.9	28,008	0.927	4	1,590	55
MC3b	3000	7,798.9	5,459	0.700	4	1,396	55
MC5a	5000	43,081.4	39,390	0.914	4	6,268	152
MC5b	5000	11,788.5	8,195	0.695	4	6,149	152
MC7a	7000	150,646.2	141,547	0.940	4	24,089	527
MC7b	7000	28,178.5	19,010	0.675	4	25,774	533

For eight different graphs, we computed the semidefinite relaxation and applied the randomized approximation procedure. For each graph, Table 1.2 shows the objective value of the semidefinite relaxation and best cut, the number of processors used to solve the problem, and the times needed to solve the SDP and apply the randomized algorithm. The processors applied the randomized procedure concurrently until the procedure had been applied 1000 times. Graphs named with an "a" have edge weights of 1 and graphs named with a "b" have edge weights of ± 1 in roughly equal proportion. As Table 1.2 shows, the quality of the cuts is high, especially when the edges have positive weights.

A randomized algorithm for stable sets [9] begins in a similar way: given a solution X^* to (MSS), find a $V \in \mathbb{R}^{(n+1)\times(n+1)}$ such that $X^* = V^T V$, select a unit vector $u \in \mathbb{R}^{n+1}$ from the unit sphere, and let $v = sign(V^T u)$. For each $(i,j) \in$ E, if $|v_i + v_j + v_{n+1}| \neq 1$, change the sign of either v_i or v_j. The stable set will be the set of vertices with the same sign as v_n. For arbitrary graphs, the constraints corresponding to the edges of the graph will be satisfied with a frequency greater than 91% [14].

From the relaxation of the graph coloring problem, a solution X^* with rank less than or equal to k identifies a legal k-coloring. More generally, Karger,

Motwani, and Sudan [37] propose a randomized algorithm that produces a k-semicoloring, an assignment of colors with relatively few adjacent vertices with the same color. We use a heuristic procedure for to obtain a legal coloring, albeit with more than k colors if necessary. For $k = 1, \ldots, n$, do the following: first let U^k be the uncolored vertices. If U^k is empty, terminate the algorithm. Sort the vertices of U^k in decreasing order of degree in $G[U^k]$, the graph induced by the uncolored vertices, and let i be the vertex with highest degree. Then build a vertex set W^k by examining vertices $j \in U^k$ in the decreasing order of X_{ij}. Add j to W^k if it is not adjacent to any of the vertices in W^k. Finally, assign the vertices in W^k color k. This algorithm is an extension of the popular algorithm proposed by Powell and Toint [49] to semidefinite programming.

The second set of tests computed the Lovász number and semidefinite relaxations of the minimum color relaxation for nine randomly generated unweighted graphs. The semidefinite relaxation of the maximum stable set problems on the complement of these graphs was also computed. The larger graphs created SDP's that were too large for a single processor. From the relaxations, feasible solutions were found by using the heuristic methods mentioned above. Like the randomized algorithm for the MAX-CUT relaxation, the processors applied the algorithm concurrently. Table 1.3 shows the results; recall that the clique size is the size of the stable set on the complement graph. In many of these problems, the bound provided by the semidefinite relaxation was significantly better than the bound provide the feasible solutions of the other combinatorial problem.

Table 1.3 Approximate Maximum-Clique and Minimum Coloring from Semidefinite Relaxations.

	Graph			Objective Values			
Name	$\|V\|$	$\|E\|$	$\|\bar{E}\|$	Clique	$\vartheta(G)$	Color	$\|P\|$
G1	100	1980	2970	9	13.1	22	16
G2	100	990	3960	17	22.1	32	16
G3	150	7823	3352	4	7.75	17	16
G4	150	5588	5587	6	12.6	26	16
G5	150	3353	7822	11	20.5	36	16
G6	200	5970	13930	5	8.8	21	32
G7	200	9950	9950	7	14.5	31	32
G8	200	13930	5970	13	23.9	46	32
G9	300	17940	26910	7	13.7	35	40

The use of cutting planes and lift-and-project methods can strengthen the semidefinite relaxations and generate optimal solutions. Yilderim [60] offers

techniques specific to the stable set problem and and Anjos [4] applies lifting procedures for the semidefinite relaxations of the maximum-cut problem. More generally, the construction of Lasserre [40, 41] and Henrion [35] uses a sequence of moment matrices and nonnegative polynomials. The work of Laurent and Rendl [42] summarizes many of these techniques.

1.7 CONCLUSIONS

Semidefinite programming provides tractable bounds and approximate solutions for several hard combinatorial optimization problems. Robust solvers have been developed that target these semidefinite programs, and some of these solvers have been implemented for parallel architectures. For interior-point methods, computing and solving the Newton equations in parallel increases the size of problems that we can solve and reduces the time needed to compute these solutions. Numerical results show that on problems of sufficient size and structure, interior-point methods exhibits good scalability on parallel architectures. The parallel version of DSDP was written to demonstrate the scalability of interior-point methods for semidefinite programming and provide computational scientists with a robust, efficient, and well-documented solver for their applications. The software is freely available from the Mathematics and Computer Science Division at Argonne National Laboratory, and we encourage its use with the terms of the license.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

We gratefully acknowledge the use of "Jazz," a Linux cluster located at Argonne's Laboratory Computing Resource Center.

References

1. F. Alizadeh. Interior-point methods in semidefinite programming with application to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

2. Miguel F. Anjos. An improved semidefinite programming relaxation for the satisfiability problem. *Mathematical Programming*, 102(3):589–608, 2005.

3. Miguel F. Anjos. An improved semidefinite programming relaxation for the satisfiability problem. *Math. Program.*, 102(3):589–608, 2005.

4. Miguel F. Anjos and Henry Wolkowicz. Strengthened semidefinite relaxations via a second lifting for the max-cut problem. *Discrete Applied Mathematics*, 119(1-2):79–106, 2002.

5. Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. on Computing*, 15(4):1054–1068, November 1986.

6. Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.3.0, Argonne National Laboratory, 2005. `http://www.mcs.anl.gov/petsc`.

7. Steven J. Benson. Parallel computing on semidefinite programs. Technical Report ANL/MCS-P939-0302, Mathematics and Computer Science Division, Argonne National Laboratory, March 2003.

8. Steven J. Benson, Lois Curfman McInnes, and Jorge J. Moré. A case study in the performance and scalability of optimization algorithms. *ACM Transactions on Mathematical Software*, 27(3):361–376, September 2001.

9. Steven J. Benson and Yinyu Ye. Approximating maximum stable set and minimum graph coloring problems with the positive semidefinite relaxation. In *Applications and Algorithms of Complementarity*, volume 50 of *Applied Optimization*, pages 1–18. Kluwer Academic Publishers, 2000.

10. Steven J. Benson and Yinyu Ye. DSDP5: Software for semidefinite programming. Technical Report ANL/MCS-P1289-0905, Mathematics and

Computer Science Division, Argonne National Laboratory, September 2005. Submitted to ACM Transactions of Mathematical Software.

11. Steven J. Benson, Yinyu Ye, and Xiong Zhang. Mixed linear and semidefinite programming for combinatorial and quadratic optimization. *Optimization Methods and Software*, 11:515–544, 1999.

12. Steven J. Benson, Yinyu Ye, and Xiong Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization*, 10(2):443–461, 2000.

13. C. Berge. *Graphs and Hypergraphs*. North–Holland, Amsterdam, 1973.

14. D. Bertsimas and Y. Ye. Semidefinite relaxations, multivariate normal distributions, and order statistics. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 3, pages 1–19. Kluwer Academic Publishers, Boston, MA, 1998.

15. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK: Users' Guide*. SIAM, 1997.

16. I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.

17. Brian Borchers. CSDP 2.3 user's guide. *Optimization Methods and Software*, 11/12(1-4):597–611, 1999.

18. Brian Borchers and Joseph Young. Implementation of a primal-dual method for SDP on a parallel architecture, September 2005.

19. P. Briggs, K. Cooper, K. Kennedy, and L. Torczon. Coloring heuristics for register allocation. In *ACM Conference on Program Language Design and Implementation*, pages 275–284. The Association for Computing Machinery, 1998.

20. Samuel Burer and Renato D.C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (series B)*, 95(2):329–357, 2003.

21. G. J. Chaitin, M. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, 1981.

22. Gregory J. Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Notices (Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, Boston, Mass.)*, 17(6):98–101, June 1982.

23. C. Choi and Y. Ye. Solving sparse semidefinite programs using the dual-scaling algorithm with an iterative solver. Working Paper, Department of Management Science, The University of Iowa, Iowa City, IA, 2000.

24. Etienne de Klerk, D. V. Pasechnik, and J. P. Warners. Approximate graph colouring and max-k-cut algorithms based on the theta function. *J. of Combinatorial Optimization*, To appear, 2005.

25. Etienne de Klerk and Hans van Maaren. On semidefinite programming relaxations of (2+p)-SAT. *Annals of Mathematics and Artificial Intelligence*, 37:285–305, 2003.

26. D. De Werra. An introduction to timetabling. *European Journal of Operations Research*, 19:151–162, 1985.

27. A. Frieze and M. Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18:61–77, 1997.

28. Andreas Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1):8–14, 1986.

29. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. W.H Freeman, San Francisco, CA, 1979.

30. M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145, 1995.

31. William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, MA, 1997.

32. Grötschel, M., Lovász, L., and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

33. M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.

34. C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal of Optimization*, 10(3):673 – 696, 2000.

35. D. Henrion and J. B. Lasserre. GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi. *ACM Transactions on Mathematical Software*, 29(2):165–194, June 2003.

36. Michael A. Heroux. An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, September 2005.

37. David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *Journal of ACM*, pages 246–265, 1998.

38. Jon Kleinberg and Michel X. Goemans. The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM Journal on Discrete Mathematics*, 11(2):196–204, May 1998.

39. M. Kočvara and M. Stingl. PENNON — a code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, 18(3):317–333, 2003.

40. J. B. Lasserre. An explicit equivalent positive semidefinite program for nonliner 0-1 programs. *SIAM J. Optimization*, 12:756–769, 2002.

41. J. B. Lasserre. SDP versus LP relaxations for polynomial programming. In *Novel Approaches to Hard Discrete Optimization*, Fields Institute Communications. American Mathematical Society, 2003.

42. Monique Laurent and F. Rendl. Semidefinite programming and integer programming. In K. Aardal, G. Nemhauser, and R. Weismante, editors, *Handbook on Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*. Elsevier, 2005.

43. L. Lovász. On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.

44. Carlo Mannino and Antonio Sassano. An exact algorithm for the maximum cardinality stable set problem. *Computational Optimization and Applications*, 3(4):243–258, 1994.

45. Renato D. C. Monteiro. First- and second-order methods for semidefinite programming. *Mathematical Programming*, B97:209–244, 2003.

46. Kazuhide Nakata, Makoto Yamashita, Katsuki Fujisawa, and Masakazu Kojima. A parallel primal-dual interior-point method for semidefinite programs using positive definite matrix completion. Technical Report Research Report B-398, Dept. Math. & Comp. Sciences, Tokyo Institute of Technology, November 2003. To appear in *Parallel Computing*.

47. Yu. E. Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.

48. S. Polijak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for 0-1 quadratic programming. *Journal of Global Optimization*, 7:51–73, 1995.

49. M. J. D. Powell and P. L. Toint. On the estimation of sparse Hessian matrices. *SIAM Journal on Numerical Analysis*, 16:1060–1074, 1979.

50. Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11/12(1-4):625–653, 1999.

51. Michael J. Todd. *Semidefinite Optimization*, volume 10, pages 515–560. Cambridge University Press, Cambridge, 2001.

52. K-C. Toh and M. Kojima. Solving some large scale semidefinite programs via the conjugate residual method. *SIAM Journal of Optimization*, 12(3):669–691, 2002.

53. K.C. Toh, M.J. Todd, and R.H. Tutuncu. SDPT3 — A Matlab software package for semidefinite programming, version 2.1. *Optimization Methods and Software*, 11:545–581, 1999.

54. Y. Tymofyeyev. A parallel bundle method for large-scale semidefinite programming. Master's thesis, Department of Mathematics & Statistics, University of Maryland, Baltimore County, 2002.

55. Hans van Maaren and Linda van Norden. Sums of squares, satisfiability and maximum satisfiability. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing*, volume 3569, pages 293–307. Springer Lecture Notes in Computer Science, 2005.

56. Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe, editors. *Handbook of Semidefinite Programming*, volume 27 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, 2000.

57. D. C. Wood. A technique for coloring a graph applicable to large scale time-tabling problems. *The Computer Journal*, 3:317–319, 1969.

58. Makoto Yamashita, Katsuki Fujisawa, and Masakazu Kojima. Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optimization Methods and Software*, 18:491–505, 2003.

59. Makoto Yamashita, Katsuki Fujisawa, and Masakazu Kojima. SDPARA: Semidefinite programming algorithm parallel version. *Parallel Computing*, 29:1053–1067, 2003.

60. E. Alper Yildirim and Xiaofei Fan. On extracting maximum stable sets in perfect graphs using lovasz's theta function. *Computational Optimization and Applications*, 32:1–30, 2005.

61. Q. Zhao, S. Karisch, F. Rendl, and H. Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problems. *J. of Combinatorial Optimization*, 2(1):95–27, 1998.

Preprint ANL/MCS-P1292-0905